Carefully Study Other Systems Before Incorporating Them into Your System

William L. Fithen, Software Engineering Institute [vita[3]]

2005-10-03

Indiscriminate delegation of function to other systems can introduce vulnerabilities.

Description

When a program with privileges invokes another program, unless specific steps are taken to the contrary, the invoked program will inherit the privileges of the invoker. When such invoked programs are not written with the expectation that they will execute with unusual privileges, there is an increased potential for vulnerability. Some common examples include

- a "restricted" program invoking some program that itself can spawn an unrestricted shell
- a setuid or setgid UNIX program invoking another command, which then executes with the setuid or setgid identity instead of the original user's identity [VU#700326, Viega 02]

Delegation to "Symbolic Link Followers"

There are probably a large number of applications that are safe with respect to traditional symlink TOCTOU[10] defects but vulnerable to a variant [Securiteam 05]. This defect exists when one program, the privileged **parent**, invokes another program, the **child**, and:

1. The **child** is designed to be run directly by an interactive user or indirectly via a script on behalf of the user (this is correct).
2. The **child** does not need to run with more privileges than the user that invokes it (this is correct).
3. The **child** program assumes that its invoker trusts all files that are specified as arguments (this is correct).
4. The **child** program follows symlinks (this is correct; it is the point of symbolic links).
5. The **parent** delegates an operation to the child with **parent**'s privileges, passing filenames as arguments (this is not correct).
6. The files named in the filenames passed by the **parent** are
    1. controllable, or predictable, by the adversary, and
    2. in a directory that's writable by the adversary.

The attacker can use a conventional symlink attack on the filename arguments that the **parent** passes to the **child**.

3. daisy:320 (Fithen, William L.)

10. daisy:332#TOCTOU (Follow the Rules Regarding Concurrency Management)

The security vulnerability lies in step 6. It is the **parent**'s responsibility to protect against this; changing the **child** could sits normal function.

Compared to conventional symlinks attacks, the attack here need not involve temporary files.

Applicable Context

All of the following must be true:

- The invoking program must be operating with unusual privileges.
- The invoking program must *spawn* another program to perform some function on its behalf.
- The spawned program must not expect to have been invoked with other than *normal* privileges.

Impacts Being Mitigated

- Impact #1:
  - **Minimally:** The minimal impact is nothing. Programs invoked by a privileged program may not perform any action that makes the delegation vulnerable.
  - **Maximally:** The maximal impact can be significant or even ultimate. If the invoked program can be influenced by the adversary to take some action from which the adversary will benefit, that action is the maximal impact of the delegation failure.

Security Policies to be Preserved

- Policy #1
  - The programmer who delegates some action to another program may be unaware of the security policy being implemented by the combination.

How to Recognize this Defect

- Look for operations that *spawn* subprocesses and determine if the invoked program is invoked with *unexpected* privileges.

Mitigation Advice

To Engineers:

- **Efficacy:** LOW
- Audit the invoked program to make sure that it cannot perform any action that violates the expected security policy.

To Engineers:

- **Efficacy:** HIGH
- Set the invoking program's privileges back to *normal* before invoking a child program.

To Engineers:

- **Efficacy:** INFINITE
- Eliminate the delegation, typically by incorporating the required function of the delegated program into the invoking program.

References

[Bishop 87]	Bishop, M. "How To Write A Setuid Program." *;login: 12*, 1 (January/February 1987): 5-11.
[SecuriTeam 05]	Beyond Security: SecuriTeam. *Second-Order Symlink Vulnerabilities*. June, 20, 2005. http://www.securiteam.com/securityreviews/5FP0L00G1E.html.
[Viega 02]	Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley, 2002.
[VU#700326]	Dormann, Will. *Vulnerability Note VU#700326: cdrecord fails to set proper permissions on programs specified in RSH environment variable*. cert.org, 09/17/2004. http://www.kb.cert.org/vuls/id/700326.

SEI Copyright

Fields

Name	Value
Copyright Holder	SEI

Fields

1. http://www.sei.cmu.edu/about/legal-permissions.html

Name	Value
is-content-area-overview	false
Content Areas	Knowledge/Guidelines
SDLC Relevance	Implementation
Workflow State	Publishable